Isabelle coursework exercises

John Wickerson

Autumn term 2022

There are four tasks, all worth the same number of marks. They appear in roughly increasing order of difficulty. The first three tasks are independent from each other, so failure to complete one task should have no bearing on later tasks. Task 4 builds on Task 3. Tasks labelled (⋆) are expected to be reasonably straightforward. Tasks labelled (⋆⋆) should be manageable but may require quite a bit of thinking, and it may be necessary to consult additional sources of information, such as the Isabelle manual and Stack Overflow. Tasks labelled (⋆⋆⋆) are more ambitious still.

Marking principles. It is not expected that students will complete all parts of all the tasks. Partial credit will be given to partial answers. If you are unable to complete a proof, partial credit will be given for explaining your thinking process in the form of `(*comments*)` in the Isabelle file.

Submission process. You are expected to produce a single Isabelle theory file called `YourName.thy`. This file should contain all of the definitions and proofs for all of the tasks below that you have attempted.

Plagiarism policy. You **are** allowed to consult the coursework tasks from previous years – the questions and model solutions for these are available. You **are** allowed to consult internet sources like Isabelle tutorials. You **are** allowed to work together with the other student in your pair. You **are** allowed to ask questions on Stack Overflow or the Isabelle mailing list, but make your questions generic (e.g. "Why isn't the `subst` method working as I expected?"); please **don't** ask for solutions to these specific tasks! And please **don't** share your answers to these tasks outside of your own pair. If you would like to share your answers to these tasks publicly, e.g. on a public GitHub repo, you are welcome to do so, but please check with me first, because some students may still be working on the coursework with an extended deadline.

Task 1 ($\star$) This task is about designing circuitry that implements binary addition, and proving that the circuitry is correct.

Consider the following `fulladder` function, which takes three Boolean inputs (`a`, `b`, and `cin`) and uses two half-adders and an OR gate to produce two Boolean outputs (`cout` and `s`):

```
fulladder (a,b,cin) = (
  let (tmp1, tmp2) = halfadder(a,b) in
  let (tmp3, s) = halfadder(cin, tmp2) in
  let cout = tmp1 | tmp3 in
  (cout, s))
```

Provide a suitable definition of `halfadder`, and then use Isabelle to prove that `fulladder` is correct, in the sense that

$$2 \times \texttt{cout} + \texttt{s} = \texttt{a} + \texttt{b} + \texttt{cin}.$$

Task 2 ($\star\star$) Prove the following theorem in Isabelle:

> **Theorem.** *Raising any natural number to its fifth power does not change its last (decimal) digit. In other words:* $n^5 \bmod 10 = n \bmod 10$.

Also prove that the theorem does not apply to *sixth* powers.

Task 3 ($\star\star$) This task builds on the `circuit` datatype from the worksheet. We shall add an extra optimisation that exploits the following Boolean identities:

$$\begin{aligned} a \vee a &\equiv a \\ a \wedge a &\equiv a \end{aligned}$$

The following function, called `opt_ident`, traverses a given circuit looking for opportunities to apply those identities (in the left-to-right direction). Each time the identity is applied, one gate is removed from the circuit, thus reducing its area.

```
fun opt_ident where
  "opt_ident (NOT c) = NOT (opt_ident c)"
| "opt_ident (AND c1 c2) = (
   let c1' = opt_ident c1 in
   let c2' = opt_ident c2 in
   if c1' = c2' then c1' else AND c1' c2')"
| "opt_ident (OR c1 c2) = (
   let c1' = opt_ident c1 in
   let c2' = opt_ident c2 in
   if c1' = c2' then c1' else OR c1' c2')"
| "opt_ident TRUE = TRUE"
| "opt_ident FALSE = FALSE"
| "opt_ident (INPUT i) = INPUT i"
```

Use Isabelle to prove that `opt_ident` is correct. That is, prove for any circuit c that `opt_ident` (c) has the same input/output behaviour as c. Also prove that `opt_ident` never increases circuit area (as measured by counting the number of gates).

Task 4 ($\star\star\star$) This task also builds on the `circuit` datatype from the worksheet.

By adapting `opt_ident` from the previous task or otherwise, implement a function called `opt_redundancy` that exploits the following Boolean identities:

$$\begin{aligned} a \vee (a \wedge b) &\equiv a \\ (a \wedge b) \vee a &\equiv a \\ a \wedge (a \vee b) &\equiv a \\ (a \vee b) \wedge a &\equiv a \end{aligned}$$

Use Isabelle to prove that `opt_redundancy` is correct. Also prove that it never increases circuit area.